General Coding Practices

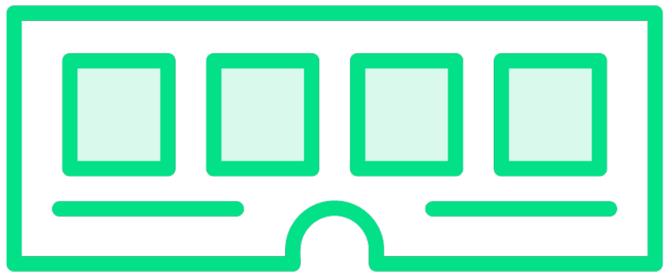
Paul Mooney
Chief Software Architect, Microsoft MVP

@daishisystems | www.insidethecpu.com

Overview

Coming Up

- Memory management and buffer boundary checking
- Avoiding Go's Unsafe package
- Manual memory deallocation
- Cross-Site Request Forgery (CSRF)
- State-changing requests, focusing on POST security
- Nonce-based validation to process valid requests
- Regular expressions in Go
- Risks of regex misuse, including ReDoS attacks
- Go's RE2 limitations, safety, and balance

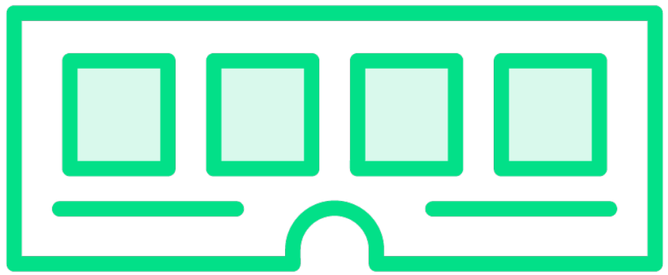


Memory Management

- Memory management for web safety
- Implementing OWASP guidelines against threats
- Protecting user input/output is essential
- Input Validation, Output Encoding modules
- Buffer boundary checking necessity
- Panic trigger and runtime error
- Go's Unsafe package
- Go's Testing package
- Garbage collector eases memory deallocation task

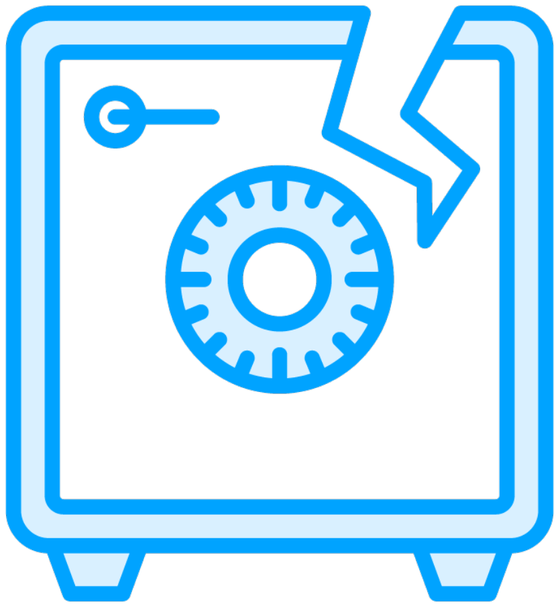
Boundary Checking

```
func main() {
    strings := []string{"aaa", "bbb", "ccc", "ddd"}
    // Our loop is not checking the MAP length -> BAD
    for i := 0; i < 5; i++ {
        if len(strings[i]) > 0 {
            fmt.Println(strings[i])
        }
    }
}
```



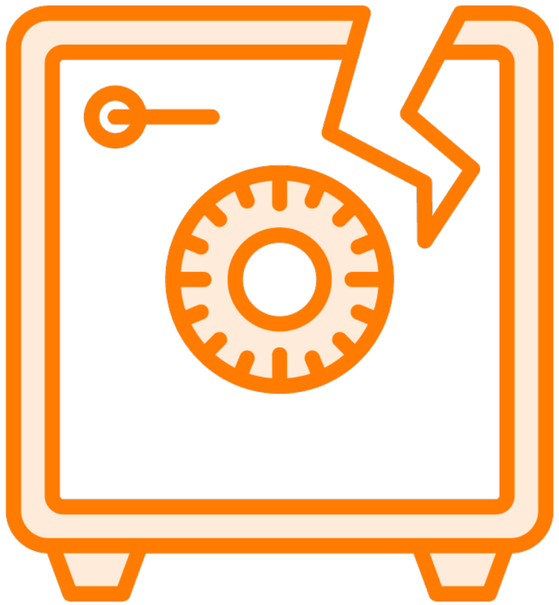
Memory Management

- Memory management for web safety
- Implementing OWASP guidelines against threats
- Protecting user input/output is essential
- Input Validation, Output Encoding modules
- Buffer boundary checking necessity
- Panic trigger and runtime error
- Go's Unsafe package
- Go's Testing package
- Garbage collector eases memory deallocation task



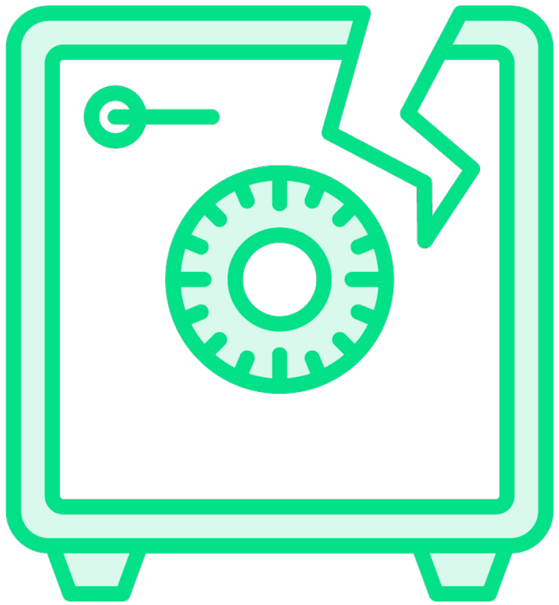
Cross-Site Request Forgery

- CSRF forces authenticated users into unwanted web app actions
- Attackers manipulate users while they're authenticated on the site
- CSRF attacks aim to change application state, not data theft
- Social engineering enables CSRF, tricking users via links in communication
- Users can be manipulated to alter critical account settings
- Module teaches understanding of CSRF, defence techniques, and robust measures



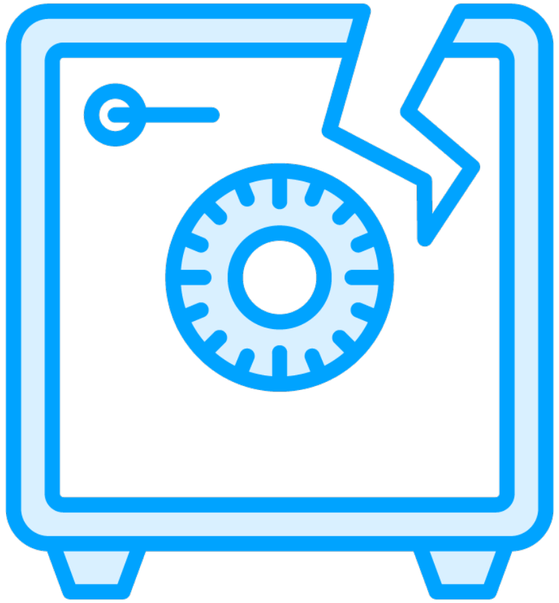
Cross-Site Request Forgery: The Challenge

- Changing HTTP verbs, using secret cookies inadequate
- Secret cookies, URL rewriting, HTTPS
- These measures alone don't counteract CSRF
- CSRF attacks exploit server limitations
- Servers struggle to differentiate requests
- Legitimate and malicious requests seem similar



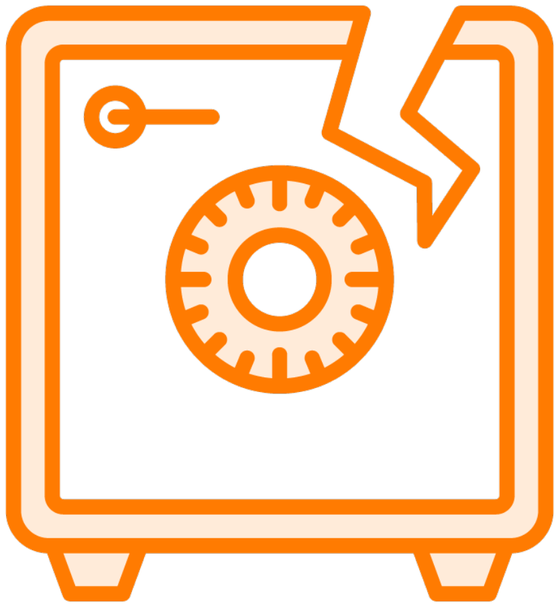
Cross-Site Request Forgery: The Solution

- Mitigate CSRF attacks by targeting state-changing POST requests
- Implement "nonce-based token validation" to address CSRF effectively
- Server generates a unique nonce for each form page
- Nonce is included as a hidden field in the form
- Server validates the token's authenticity before processing the request
- Only requests accompanied by a verified, valid token are accepted



Ensuring Effectiveness

- Assign a unique nonce/token per user session
- Generate a large, random value for the nonce/token
- Use a cryptographically secure random number generator for generation



Best Practices

- Poor programming practices can lead to state-changing GET requests
- PUT and DELETE methods are common targets for CSRF attacks
- Implement CSRF protection measures for API endpoints
- Web Application Frameworks provide built-in CSRF solutions
- Consider adopting a framework for robust CSRF protection
- Frameworks offer battle-proven CSRF prevention mechanisms

Regular Expressions

- Regular expressions validate user input in web applications
- Mastering regular expressions requires careful understanding
- Go language integrates the secure RE2 package
- RE2 ensures linear-time performance and graceful failure
- Regular Expression Denial of Service (ReDoS) explained
- Excessive evaluation time caused by improper implementation

Validate Email Address

```
^([a-zA-Z0-9])(([-.]|[_]+)?([a-zA-Z0-9]+))*(@){1}[a-z0-9]+[.]{1}(([a-z]{2,3})|([a-z]{2,3}[.]{1}[a-z]{2,3}))$
```

RE2 Safety Mechanisms

- Go's regex engine RE2 consciously avoids backtracking
- Backreferences and Lookaround features are not supported in RE2
- Opening and closing tag names must correspond
- There may optionally be text between tags
- “.*?” suffices
- Tag closing hinges on what was identified as the opening tag

Alternative Solutions

- Develop workarounds for limitations in Go's regex features
- Consider dlclark/regexp2 for more fully featured alternative
- Balancing functionality and safety in regex choice

Summary

General Coding Practices

- Memory management complexities in Go
- Buffer boundary checks, potential errors
- Go's Unsafe package, manual deallocation
- Cross-Site Request Forgery
- Implemented CSRF protections, nonce-based validation
- Examined regular expressions, RE2 features, safety balance